UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/726,067	12/01/2003	Matthew Balint	50235-0840	2564

29989 7590 11/01/2007

HICKMAN PALERMO TRUONG & BECKER, LLP
2055 GATEWAY PLACE
SUITE 550
SAN JOSE, CA 95110

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
11/01/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/726,067	BALINT ET AL.	
	Examiner	Art Unit	
	Ben C. Wang	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *11 July 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-78* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-78* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *7/6/2007, 8/6/2007, 9/20/2007*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated July 11, 2007, responding to the Office action mailed April 20, 2007 provided in the rejection of claims 1-78, wherein claims 1-3, 5-7, 9, 11-15, 18-21, 24-29, 31-54, 57-59, 61, 63-67, 70-73, and 76-78 are amended, claims 4, 30, and 56 are canceled.

Claims 1-78 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims rejection have been fully considered but are moot in view of the new grounds of rejection.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1-3, 5-10, 12, 14-16, 18-29, 31-36, 38, 40-42, 44-55, 57-62, 64, 66-68, and 70-78 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gary. D. Foster (Pat. No. US 6,675,382 B1) (hereinafter 'Foster') in view of Oreizy et al., (*Architecture-Based Runtime Software Evolution, 1998, IEEE*) (hereinafter 'Oreizy')

3. **As to claim 1** (Currently Amended), Foster discloses a method of dynamic installation and activation of software packages in a node in a distributed network of nodes, the method comprising the computer-implemented steps of:

- Storing, in a software package storage of a master node in the distributed network, a plurality of software packages and a plurality of software modules (e.g., Col. 12, Lines 43-52 – the software files required for installation may be directly downloaded from the remote server onto the local client system; a set of database files to track information pertaining to software packages that have been installed or distributed) that the nodes in the distributed network will be using (e.g., Col. 10, Line 64 through Col. 11, Line 4 – an older release);
- wherein each software package of the plurality of software packages (e.g., Fig. 2 – Package; Col. 6, Lines 37-41) contains at least one module (e.g., Col. 7, Lines 1-9 – payload file contains all files that are required for the installation of computer software) and associated dependency information (e.g., Col. 6, Lines 60-64 – a control file that contains control information pertaining to those files and their dependencies);
- receiving a software update for a node on said master node (e.g., Col. 12, Lines 43-45);
- wherein the software update contains a set of one or more software packages (e.g., Abstract, Lines 1-5; Col. 3, Lines 46-51; Fig. 2 – Package; Col. 6, Lines 37-41);

- storing the software update (e.g., Col. 12, Lines 43-45) on said software package storage (e.g., Col. 12, Lines 43-45);
- wherein said master node passes said node identities of one or more software packages to be updated (i.e., Col. 8, Lines 34-36) and module dependencies (e.g., Col. 8, Lines 22-24);

Further, Foster discloses a method and apparatus for packing and distributing software (e.g., Abstract, Lines 1-2) and check dependencies (e.g., Fig. 4 – element 410 "Check Dependencies"; Col. 9, Lines 24-26 – At step 410, prior to installing package 200, any dependencies are checked as specified by...), but does not explicitly disclose that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested.

However, in an analogous art of *Architecture-Based Runtime Software Evolution*, Oreizy discloses:

- said master node notifies said node that a software update is being requested (e.g., Sec. 3 – Previous Approaches to Runtime Change, 6th Par. – Kramer and Magee present "a structural-based approach to runtime change of a distributed system's configuration. In their approach, a configuration consists of processing nodes interconnected using bidirectional communication links. When a runtime change is required, a reconfiguration manager (master node) orders (notifies) processing nodes directly affected by the change and nodes directly adjacent to

them ....; This ensures that nodes directly affected by a change will not receive service requests during the course of the change" (emphasis added)); and

- said node determines, using the module dependencies, running processes on said node that will be affected by the software update (e.g., Abstract, 4th Par. – "A distinctive feature of software architectures is the explicit modeling of connector. Connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution"; Sec. 6 – Applying Concepts to a Specific Architectural Style, 2nd Par. – "In the C2-style, all communication among components occurs via connectors, thus minimizing component interdependencies and strictly separating computation from communication. The style also imposes topological constraints: every component has a 'top' and a 'bottom' side, with a single communication port on each side. This restriction greatly simplifies the task of adding, removing, or reconnecting a component" (emphasis added)).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Oreizy into the Foster's system to further provide that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested in Foster system.

The motivation is that it would advantageously enhance the Foster's system by taking, advancing and/or incorporating Oreizy's system which provides the software

architectures which explicitly models connectors that mediate and govern interactions among components, and thereby separate computation form communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution; the benefits of runtime evolution are from a systematic, principled approach to runtime change supported by a reusable infrastructure as once suggested by Oreizy (i.e., Sec. of Introduction, 1st Para., 2nd Para., 4th Para.).

4. **As to claim 2** (incorporating the rejection in claim 1) (Currently Amended), Foster discloses a method wherein each module has a binary signature (e.g., Fig. 2, element 230 – digital signature file; Col. 11, Lines 61 through Col. 12, Line 10).

5. **As to claim 3** (incorporating the rejection in claim 1) (Currently Amended), Foster discloses a method wherein each node has a list of desired characteristics stored on said master node which is compared by said master node to each module in the software update to determine which modules in the set of one or more software packages should be sent to a node (i.e., Col. 7, Lines 35-38 – OSVERSION and PLATFORM fields; Col. 8, Lines 34-36; Col. 11, Lines 28-48).

6. **As to claim 5** (incorporating the rejection in claim 1) (Currently Amended), Oreizy discloses a method wherein said node notifies affected processes that the software update is being requested; wherein each notified process evaluates the effect that the software update will have on its operation; wherein if any of the notified

processes determine that the software update will degrade or have a negative impact on said node's normal operation, the process returns a veto to said node; and wherein if a process finds that the software update will have no negative effects, the process returns an acceptance of the software update to said node (e.g., Sec. 1 - Introduction, 4th Para. – connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies,...; Sec. 4.3 – Runtime Component Replacement, 2nd Para. – the model rejects upgraded components when they do not satisfy explicit performance and accuracy requirement).

7. **As to claim 6** (incorporating the rejection in claim 5) (Currently Amended), Oreizy discloses a method wherein said node waits for all of the notified processes to return results of their evaluations and once all of the processes have reported to said node, said node notifies said master node if any of the processes have vetoed the software update (e.g., Sec. 1 - Introduction, 4th Para. – connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies,...; Sec. 4.3 – Runtime Component Replacement, 2nd Para. – the model rejects upgraded components when they do not satisfy explicit performance and accuracy requirement).

8. **As to claim 7** (incorporating the rejection in claim 6) (Currently Amended), Foster discloses a method wherein if said master node receives an acceptance from

said node then said master node sends the set of one or more software packages (e.g., Col. 12, Lines 43-45 – the software files required for installation may be directly downloaded from the remote server onto the local client system) for the software update from said software package storage means (e.g., Col. 12, Lines 43-45 – the software files required for installation may be directly downloaded from the remote server onto the local client system) to said node (e.g., Col. 12, Lines 43-45).

9. **As to claim 8** (incorporating the rejection in claim 7) (Original), Oreizy discloses a method wherein said node immediately runs software package modules, by loading the modules from the software package(s) and signals processes that are being replaced by the modules and the affected processes that the changeover is going to occur; wherein when all of the signaled processes indicate that they are ready and waiting for the changeover, said node starts new modules and signals the affected processes that the changeover has occurred; wherein each module starts without affecting normal operation of said node; and wherein each affected process restarts, if required, without affecting normal operation of said node (e.g., Sec. 1 - Introduction, 4th Para. – connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies,...; Sec. 4.3 – Runtime Component Replacement, 2nd Para. – the model rejects upgraded components when they do not satisfy explicit performance and accuracy requirement; Sec. 4.3 – Runtime Component Replacement, 1st Para. – the state of the executing component must be transferred to the new component, and both components must not

be simultaneously active during the change; corrective and adaptive evolution are characteristic of such changes; Sec. 5.2 – Connectors. 4th Para.).

10. **As to claim 9** (incorporating the rejection in claim 8) (Currently Amended), Foster discloses a method wherein said node continues with normal operations and notifies said master node that it has completed the software update; and wherein said master node checks the module dependencies to ensure that any inter-nodal and intra-node dependencies are complete (e.g., Fig. 4, step – Check Dependencies; Col. 9, Lines 18-33).

11. **As to claim 10** (incorporating the rejection in claim 9) (Original), Foster discloses a method wherein if there are any discrepancies in the inter-nodal and intra-node dependencies (e.g., Fig. 4, step 410 – Check Dependencies; Col. 8, Lines 27-29), then said master node notifies a user (e.g., Fig.4, step 425; Col. 9, Lines 23-28; Col. 10, Lines 10-14).

12. **As to claim 12** (incorporating the rejection in claim 7) (Currently Amended), Foster discloses a method wherein said node extracts version information (e.g., Col. 8, Lines 17-21) and dependency information (e.g., Col. 8, Lines 22-29) from the set of one or more software packages and stores the information in its local persistent storage (e.g., Fig. 1, element 112 – Mass Storage; Col. 6, Lines 15-19).

13. **As to claim 14**, please refer to claim **8** above accordingly.

14. **As to claims 15-16**, please refer to claims **9-10** above accordingly.

15. **As to claim 18** (incorporating the rejection in claim 6) (Currently Amended), Oreizy discloses a method wherein if said master node receives a veto from said node, then said master node does not update said node (e.g., Sec. 1 - Introduction, 4th Para. – connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies,...; Sec. 4.3 – Runtime Component Replacement, 2nd Para. – the model rejects upgraded components when they do not satisfy explicit performance and accuracy requirement) and notifies a user that the software update will adversely affect said node (e.g., Sec. 7 – Tools Supporting Architecture-Based Evolution of Software System, sub-sec of Describing Runtime Change, 2nd Para., Lines 12-14).

16. **As to claim 19** (incorporating the rejection in claim 18) (Currently Amended), Foster discloses a method further comprising receiving an indication to continue updating said node, wherein said master node forces said node to accept the software update (e.g., Col. 10, Lines 40-47).

17. **As to claim 20** (incorporating the rejection in claim 1) (Currently Amended), Foster discloses a method further comprising initiating the software update by receiving

an image containing the software update onto said master node (e.g., Col. 1, Lines 30-34).

18. **As to claim 21** (incorporating the rejection in claim 20) (Currently Amended), Foster discloses a method receiving an indication of a set of nodes and a set of software packages that are to be updated (e.g., Fig. 2 – Package; Col. 6, Lines 37-41).

19. **As to claim 22** (incorporating the rejection in claim 1) (Original), Foster discloses a method wherein the software update contains a list of nodes to be updated (e.g., Col. 1, Lines 7-8, 25-30).

20. **As to claim 23** (incorporating the rejection in claim 1) (Original), Foster discloses a method wherein the software update contains a list of software packages destined for each node (e.g., Col. 3, Lines 46-51; Fig. 2 – Package; Col. 6, Lines 37-41).

21. **As to claim 24** (incorporating the rejection in claim 1) (Currently Amended), Foster discloses a method wherein the master node has an ability to categorize nodes into classes where all of the nodes in a particular class of nodes have a same software configuration (i.e., Col. 8, Lines 34-38 - OSVERSION) and may have differing processor types (i.e., Col. 8, Lines 34-38 - PLATFORM).

22. **As to claim 25** (incorporating the rejection in claim 1) (Currently Amended), Foster discloses a method wherein a software package of the plurality of software packages contains version information (e.g., Col. 8, Lines 17-21), dependency information (e.g., Col. 8, Lines 22-24), and other metadata information pertaining to software in the package (Col. 8, Lines 12-55).

23. **As to claim 26** (incorporating the rejection in claim 25) (Currently Amended), Foster discloses a method wherein the other metadata information includes a list of application program interface (API) providers and consumers (e.g., Col. 8, Lines 30-32 – MAINTAINER field; Col. 10, Lines 56-60).

24. **As to claim 27** (Currently Amended), Foster discloses a computer-readable storage medium carrying one or more sequences of instructions for dynamic installation and activation of software packages in a node in a distributed network of nodes, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

- Storing, in a software package storage of a master node in the distributed network, a plurality of software packages and a plurality of software modules (e.g., Col. 12, Lines 43-52 – the software files required for installation may be directly downloaded from the remote server onto the local client system; a set of database files to track information pertaining to software packages that have

been installed or distributed) that the nodes in the distributed network will be using (e.g., Col. 10, Line 64 through Col. 11, Line 4 – an older release);

- wherein each software package of the plurality of software packages (e.g., Fig. 2 – Package; Col. 6, Lines 37-41) contains at least one module (e.g., Col. 7, Lines 1-9 – payload file contains all files that are required for the installation of computer software);
- receiving a software update for a node on said master node (e.g., Col. 12, Lines 43-45);
- wherein the software update contains a set of one or more software packages (Abstract, Lines 1-5; Col. 3, Lines 46-51; Fig. 2 – Package; Col. 6, Lines 37-41);
- storing the software update (e.g., Col. 12, Lines 43-45) on said software package storage (e.g., Col. 12, Lines 43-45);
- wherein said master node passes said node identities of one or more software packages to be updated (i.e., Col. 8, Lines 34-36) and module dependencies (e.g., Col. 8, Lines 22-24).

Further, Foster discloses a method and apparatus for packing and distributing software (e.g., Abstract, Lines 1-2) and check dependencies (e.g., Fig. 4 – element 410 "Check Dependencies"; Col. 9, Lines 24-26 – At step 410, prior to installing package 200, any dependencies are checked as specified by...), but does not explicitly disclose that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested.

However, in an analogous art of *Architecture-Based Runtime Software Evolution*, Oreizy discloses:

- said master node notifies said node that a software update is being requested (e.g., Sec. 3 – Previous Approaches to Runtime Change, 6th Par. – Kramer and Magee present "a structural-based approach to runtime change of a distributed system's configuration. In their approach, a configuration consists of processing nodes interconnected using bidirectional communication links. When a runtime change is required, a reconfiguration manager (master node) orders (notifies) processing nodes directly affected by the change and nodes directly adjacent to them ….; This ensures that nodes directly affected by a change will not receive service requests during the course of the change" (emphasis added)); and
- said node determines, using the module dependencies, running processes on said node that will be affected by the software update (e.g., Abstract, 4th Par. – "A distinctive feature of software architectures is the explicit modeling of connector. Connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution"; Sec. 6 – Applying Concepts to a Specific Architectural Style, 2nd Par. – "In the C2-style, all communication among components occurs via connectors, thus minimizing component interdependencies and strictly separating computation from communication. The style also imposes topological constraints: every component has a 'top' and a 'bottom' side, with a single communication port on

each side. This restriction greatly simplifies the task of adding, removing, or reconnecting a component" (emphasis added)).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Oreizy into the Foster's system to further provide that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested in Foster system.

The motivation is that it would advantageously enhance the Foster's system by taking, advancing and/or incorporating Oreizy's system which provides the software architectures which explicitly models connectors that mediate and govern interactions among components, and thereby separate computation form communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution; the benefits of runtime evolution are from a systematic, principled approach to runtime change supported by a reusable infrastructure as once suggested by Oreizy (i.e., Sec. of Introduction, 1st Para., 2nd Para., 4th Para.).

25. **As to claims 28-30**, please refer to claims **2-4** above accordingly.

26. **As to claims 31-32**, please refer to claims **5-6** above accordingly.

27. **As to claim 33**, please refer to claim **7** above accordingly.

28. **As to claim 34**, please refer to claim **8** above accordingly.

29. **As to claims 35-36**, please refer to claims **9-10** above accordingly.

30. **As to claim 38**, please refer to claim **12** above accordingly.

31. **As to claim 40**, please refer to claim **8** above accordingly.

32. **As to claims 41-42**, please refer to claims **9-10** above accordingly.

33. **As to claim 44**, please refer to claim **18** above accordingly.

34. **As to claims 45-46**, please refer to claims **19-20** above accordingly.

35. **As to claims 47-52**, please refer to claims **21-26** above accordingly.

36. **As to claim 53** (Currently Amended), an apparatus, comprising:

- a master node (e.g., Fig. 1, element 126 – Server; Col. 6, Lines 8-19 – remote server computer might transmit a requested code for an application program through internet, local network and communication interface);
- means for storing, in a software package storage of the master node, a plurality of software packages and as plurality of software modules (e.g., Col. 12, Lines

43-52 – the software files required for installation may be directly downloaded from the remote server onto the local client system; a set of database files to track information pertaining to software packages that have been installed or distributed) that nodes in a distributed network will be using (e.g., Col. 10, Line 64 through Col. 11, Line 4 – an older release);

- wherein each software package of the plurality of software packages contains (e.g., Fig. 2 – Package; Col. 6, Lines 37-41) at least one module (e.g., Col. 7, Lines 1-9 – payload file contains all files that are required for the installation of computer software); means for receiving a software update for a node on said master node (e.g., Col. 12, Lines 43-45);
- wherein the software update contains a set of one or more software packages (e.g., Abstract, Lines 1-5; Col. 3, Lines 46-51; Fig. 2 – Package; Col. 6, Lines 37-41);
- means for storing the software update (e.g., Col. 12, Lines 43-45) on said software package storage (e.g., Col. 12, Lines 43-45);
- wherein said master node passes said node identities of one or more software packages to be updated (i.e., Col. 8, Lines 34-36) and module dependencies (e.g., Col. 8, Lines 22-24).

Further, Foster discloses a method and apparatus for packing and distributing software (e.g., Abstract, Lines 1-2) and check dependencies (e.g., Fig. 4 – element 410 "Check Dependencies"; Col. 9, Lines 24-26 – At step 410, prior to installing package 200, any dependencies are checked as specified by…), but does not explicitly disclose

that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested.

However, in an analogous art of *Architecture-Based Runtime Software Evolution*, Oreizy discloses:

- said master node notifies said node that a software update is being requested (e.g., Sec. 3 – Previous Approaches to Runtime Change, 6th Par. – Kramer and Magee present "a structural-based approach to runtime change of a distributed system's configuration. In their approach, a configuration consists of processing nodes interconnected using bidirectional communication links. When a runtime change is required, a reconfiguration manager (master node) orders (notifies) processing nodes directly affected by the change and nodes directly adjacent to them ...; This ensures that nodes directly affected by a change will not receive service requests during the course of the change" (emphasis added)); and
- said node determines, using the module dependencies, running processes on said node that will be affected by the software update (e.g., Abstract, 4th Par. – "A distinctive feature of software architectures is the explicit modeling of connector. Connectors mediate and govern interactions among components, and thereby separate computation from communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution"; Sec. 6 – Applying Concepts to a Specific Architectural Style, 2nd Par. – "In the C2-style, all communication among components occurs via connectors, thus

minimizing component interdependencies and strictly separating computation from communication. The style also imposes topological constraints: every component has a 'top' and a 'bottom' side, with a single communication port on each side. This restriction greatly simplifies the task of adding, removing, or reconnecting a component" (emphasis added)).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Oreizy into the Foster's system to further provide that said node determines, using the module dependencies, running processes on said node that will be affected by the software update and said master node notifies said node that a software update is being requested in Foster system.

The motivation is that it would advantageously enhance the Foster's system by taking, advancing and/or incorporating Oreizy's system which provides the software architectures which explicitly models connectors that mediate and govern interactions among components, and thereby separate computation form communication, minimize component interdependencies, and facilitate system understanding, analysis, and evolution; the benefits of runtime evolution are from a systematic, principled approach to runtime change supported by a reusable infrastructure as once suggested by Oreizy (i.e., Sec. of Introduction, 1st Para., 2nd Para., 4th Para.).

37. **As to claims 54-56**, please refer to claims **2-4** above accordingly.

38. **As to claims 57-58**, please refer to claims **5-6** above accordingly.

39. **As to claim 59**, please refer to claim **7** above accordingly.

40. **As to claim 60**, please refer to claim **8** above accordingly.

41. **As to claims 61-62**, please refer to claims **9-10** above accordingly.

42. **As to claim 64**, please refer to claim **12** above accordingly.

43. **As to claim 66**, please refer to claim **8** above accordingly.

44. **As to claims 67-68**, please refer to claims **9-10** above accordingly.

45. **As to claim 70**, please refer to claim **18** above accordingly.

46. **As to claims 71-72**, please refer to claims **19-20** above accordingly.

47. **As to claims 73-78**, please refer to claims **21-26** above accordingly.

48. Claims 11, 13, 17, 37, 39, 43, 63, 65, and 69 are rejected under 35 U.S.C. 103(a) as being unpatentable over Foster in view of Oreizy and in further view of Moshir et al., (Pub. No. US 2004/0003266 A1) (hereinafter 'Moshir')

49. **As to claim 11** (incorporating the rejection in claim 8) (Currently Amended), Foster and Oreizy do not explicitly disclose a method further comprising storing, in the software package storage, older versions of the software packages and the software modules that are kept for regressing said node back to a previous module or software package version, wherein when said node does not store the set of one or more software packages in its local persistent storage, then said node can later regress back to previous modules stored in the local persistent storage if it restarts or said master node tells it to regress.

However, in an analogous art of *non-invasive automatic offsite patch fingerprinting and updating system and methods*, Moshir discloses a method further comprising storing, in the software package storage, older versions of the software packages and the software modules that are kept for regressing said node back to a previous module or software package version, wherein when said node does not store the set of one or more software packages in its local persistent storage, then said node can later regress back to previous modules stored in the local persistent storage if it restarts or said master node tells it to regress (e.g., Abstract, Lines 6-9 – when a failure is detected, the rollout is stopped and the software can be automatically removed from those computers that already were updated; [0019], Lines 4-7; [0030], Lines 6-13 – if the package has been installed on more than one computer, they all can be removed).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moshir into the Foster-

Oreizy's system to further provide a method further comprising storing, in the software package storage, older versions of the software packages and the software modules that are kept for regressing said node back to a previous module or software package version, wherein when said node does not store the set of one or more software packages in its local persistent storage, then said node can later regress back to previous modules stored in the local persistent storage if it restarts or said master node tells it to regress in Foster-Oreizy system.

The motivation is that it would enhance the Foster-Oreizy's system by taking, advancing and/or incorporating Moshir's system which facilitates software deployment, software installation, software updating.... Across multiple operating systems and devices, across a network as once suggested by Moshir (i.e., [0020]).

50. **As to claim 13** (incorporating the rejection in claim 12) (Currently Amended), Foster discloses digital signature file (e.g., Fig. 2, element 230 – Digital Signature File) but both Foster and Oreizy do not explicitly disclose method wherein said node compares binary signatures of modules in the set of one or more software packages with corresponding modules stored in the local persistent storage to discover which modules have been updated; wherein any binary signatures that match indicate that the module has not changed; and wherein any modules that have different binary signatures replace the corresponding modules stored in the local persistent storage.

However, in an analogous art of *non-invasive automatic offsite patch fingerprinting and updating system and methods*, Moshir discloses method wherein said

node compares binary signatures of modules in the set of one or more software packages with corresponding modules stored in the local persistent storage to discover which modules have been updated; wherein any binary signatures that match indicate that the module has not changed; and wherein any modules that have different binary signatures replace the corresponding modules stored in the local persistent storage (e.g., Fig. 9, elements 908 – Existence Test , 910 – Signature Block; [0090] – an existence test which can use the signature block information to determine if a specific patch has been loaded on a machine; [0092]-[0093]; [0106]-[0107]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moshir into the Foster-Oreizy's system to further provide method wherein said node compares binary signatures of modules in the set of one or more software packages with corresponding modules stored in the local persistent storage to discover which modules have been updated; wherein any binary signatures that match indicate that the module has not changed; and wherein any modules that have different binary signatures replace the corresponding modules stored in the local persistent storage in Foster-Oreizy system.

The motivation is that it would enhance the Foster-Oreizy's system by taking, advancing and/or incorporating Moshir's system which facilitates software deployment, software installation, software updating.... Across multiple operating systems and devices, across a network as once suggested by Moshir (i.e., [0020]).

51. **As to claim 17** (incorporating the rejection in claim 6) (Original), Foster and Oreizy do not explicitly disclose a method wherein if more than one node was being updated, the software update will not occur if any node vetoes the software update.

However, in an analogous art of *non-invasive automatic offsite patch fingerprinting and updating system and methods*, Moshir discloses a method wherein if more than one node was being updated, the software update will not occur if any node vetoes the software update (e.g., Abstract, Lines 6-9 – when a failure is detected, the rollout is stopped and the software can be automatically removed from those computers that already were updated; [0019], Lines 4-7; [0030], Lines 6-13 – if the package has been installed on more than one computer, they all can be removed).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moshir into the Foster-Oreizy's system to further provide a method wherein if more than one node was being updated, the software update will not occur if any node vetoes the software update in Foster-Oreizy system.

The motivation is that it would enhance the Foster-Oreizy's system by taking, advancing and/or incorporating Moshir's system which facilitates software deployment, software installation, software updating.... Across multiple operating systems and devices, across a network as once suggested by Moshir (i.e., [0020]).

52. **As to claim 37**, please refer to claim **11** above accordingly.

53. **As to claim 39**, please refer to claim **13** above accordingly.

54. **As to claim 43**, please refer to claim **17** above accordingly.

55. **As to claim 63**, please refer to claim **11** above accordingly.

56. **As to claim 65**, please refer to claim **13** above accordingly.

57. **As to claim 69**, please refer to claim **17** above accordingly.

Conclusion

58. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW

September 25, 2007